I J C T A, 8(5), 2015, pp. 1911-1916

Cluster Computing Paradigms– A Comparative study of Evolving Frameworks

N. Anila Sundar[1], Vijay Krishna Menon[2] and P.N. Kumar[3]

Abstract: Cluster computing is an approach for storing and processing huge amount of data that is being generated. Hadoop and Spark are the two cluster computing platforms which are prominent today. Hadoop incorporates the MapReduce concept and is scalable as well as fault-tolerant. But the limitations of Hadoop paved way for another cluster computing framework named Spark. It is faster and can also manage multiple workloads due to its in-memory processing. In this paper, we discuss the underlying concepts of Hadoop and mention the limitations that led to the development of Spark. Further we give a detailed description about Spark framework and its advantages. We demonstrate a wordcount problem in both Hadoop and Spark and do a comparative study.

Index Terms: Spark; Hadoop; RDD; MapReduce; cluster computing

1. INTRODUCTION

Over the last few years there has been an exponential increase in the volume of data that is being generated. It refers to the massive and complex data that is being generated by users. In the past, few companies were responsible for generating the data, while in the present scenario users are generating the data and they are consuming as well. The huge amount of data, if processed and analyzed can generate useful information that would be helpful in many fields such as health care, business, signal processing and many more. In 2004, Google released a paper on MapReduce which came up with a new way of dealing with big data. It addressed the limitations of cluster computing and included two aspects: a MapReduce engine (mapred) and a distributed fault tolerant file system (HDFS). Soon it became popular and was being used for a number of applications, it was found inefficient for three types of applications, viz. Interactive queries, iterative jobs and online processing.

The need to address these limitations leads to the development of Spark. Spark is a cluster computing framework for fast and large scale data processing. This framework has features which eliminates the limitations of Hadoop, as well as maintains the property of fault-tolerance and scalability. Further we carry out an in-depth study of the concepts underlying Spark and compare it with Hadoop framework with the aid of a toy problem such as word count program.

2. CLUSTER COMPUTING

Cluster computing can be described as a fusion of the fields of parallel, high-performance, distributed and high-availability computing [3]. Cluster can be defined as a group of loosely coupled computers that work together in order to complete some task. In the current scenario, managing a cluster environment is considered a big challenge because of its increased complexity and size. The various projects for maintaining the cluster environment are Apache Ambari, Apache Mesos, Plaform MapReduce and Zettaset Orchestrator.

[1,2,3] Department of CSE, Amrita Vishwa Vidyapeetham, Coimbatore-641112, India, *Email: anila612@gmail.com, m_vijaykrishna@cb.amrita.edu, pn_kumar@cb.amrita.edu*

While in Spark, the cluster manager that is in-built is called the DAGScheduler and it can easily run over a variety of cluster managers, including Hadoop YARN and Apache Mesos.

2.1. Hadoop

Hadoop is an open source distributed parallel computing platform, which is mainly composed of MapReduce algorithm and a distributed file system [5]. Essentially it accomplishes two tasks: massive data storage and faster processing. Hadoop addressed the problems of cluster computing by storing data redundantly among the machines and moving the computation closer to the data. At its core, Hadoop has two major layers: Storage layer (HDFS) and Processing and computation layer (MapReduce). Hadoop Distributed File System (HDFS) specifies how the huge amount of data will be stored over the computers. In HDFS file system the data is stored redundantly among the nodes. HDFS has master/slave architecture [5].

MapReduce refers to two separate and distinct tasks that Hadoop programs perform. The first is the Map job, which takes a set of data, converts it into another set of data, where individual elements are broken down into tuples (key/value pairs) [4]. These key value pair are grouped with respect to the common key. The Reduce job takes the output from a map as input and combines the data tuples into a set of values Hadoop framework gained popularity because it was fault-tolerant and scalable. But when the framework was used for various workloads it was realized that for various scenarios MapReduce is deficient [1] for the following:

- Iterative jobs: A function is applied repeatedly to the same dataset to optimize a parameter. Because the data is stored on the disk, accessing it again and again is complex and slow.
- Interactive jobs: In Hadoop querying is done by SQL interface like Pig/Hive. Each query will run a separate MapReduce job and read data from the disk.
- Online Streaming: Hadoop was initially designed for batch processing, which means it can take a large dataset as input all at once, process it and write back a large output. The very concept of MapReduce is geared towards batch and not real-time.
- Lack of selective access to data: In order to perform any task the data to the job is consumed in a brute force manner. It scans the entire input to perform maps-side processing and may also initiate map tasks on all the input partitions. But in various cases accessing a subset of input data would be sufficient for carrying out the task. Lack of selective access will also cause high communication cost.
- Lack of early termination: For specific type of queries, examining the entire input data makes sense but for other kind of queries only a subset of the input data is sufficient to produce correct result. Early termination allows map tasks to stop processing when a specific condition holds.
- Redundant processing and recomputation: Often multiple MapReduce jobs are initiated at over-lapping times and this may result in multiple jobs doing the same processing over the same data, which is also referred as redundant processing. In MapReduce there is no way for reusing the previously computed results. Future jobs may sometimes require those results, but in MapReduce everything has to be recomputed.

2.2. Spark

Apache Spark is an open source cluster computing platform that eliminates the limitations of Hadoop while providing similar scalability and fault tolerance properties [1]. Spark extends the MapReduce model by supporting more types of computations, including interactive queries and stream processing bringing in concepts like in-memory storage and Resilient Distributed Dataset (RDD). A RDD is a read-only collection

of objects partitioned across a set of machines and can be rebuilt if a partition is lost [2]. One of the main features that Spark is offering is speed. This high speed computation is possible because of its ability to do computations in memory.

In Spark rather than replicating the data, Resilient Distributed Datasets (RDD) maintains the property of fault-tolerance. An RDD is the main abstraction of Spark which is a read only partitioned collection of records. It is a distributed collection of records which can be stored in the volatile memory and can be transformed into some other RDD through operations [2]. Each RDD is split into partitions and are stored and computed at different nodes of the cluster. RDDs can contain any type of Python, Java or Scala objects, including user-defined classes. The different ways of creating a RDD are:-

- Loading an external data set
- By parallelizing a scala collection in the driver program. The driver program is written by the developer. It launches various operations in parallel and implement high level control flow.
- By changing the persistence of an existing RDD, the cache action specifies that the result should be kept in the cache for future use and the save action writes the result back to the filesystem.RDD supports two types of operations:-
  - Transformations. Transformations are operations on RDD that results in the creation of another RDD. Map and filter are some transformation operations.
  - Action. They are the operations that return a final value to the driver program or write data to an external storage system.
- The two most important features of RDD are-
  - Lazy evaluation: This is one of the most important features of RDD which makes sure that the memory is not wasted unnecessarily. When a transformation is called through an RDD, the operation is not performed immediately. Spark internally records a metadata which specifies that these operations have been requested. Only when an action is called, actual output is generated. Thus lazy evaluation is used to reduce the number of passes it has to take over the data by grouping the operations together [6].
  - Lineage: In Spark the dataset is not replicated and stored but still it is able to provide the property of fault-tolerance. In Spark if one RDD is lost it can be recomputed back by the information that

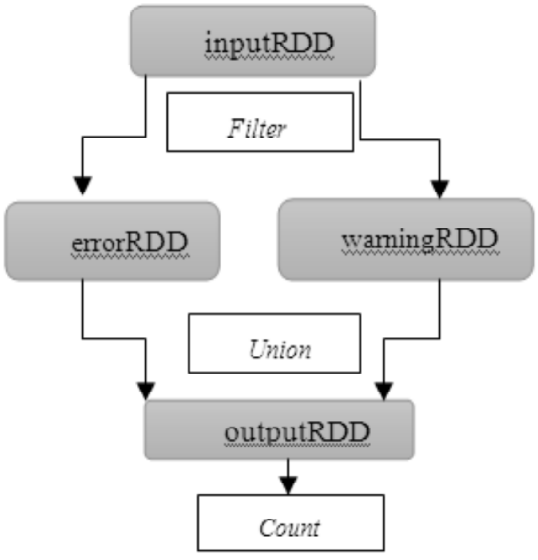

Figure 1: Lineage graph[6]

it maintains during the creation of RDD. This is done with the help of alineage graph [6]. A lineage graph keeps track of the dependencies between different RDDs. It uses this graph to compute each RDD and recover lost data. A lineage graph keeps track of the dependencies between different RDDs. It uses this graph to compute each RDD and recover lost data. Figure 1 shows a lineage graph where inputRDD is the RDD which is created when we load the dataset. Then three transformation operations are performed which results in the creation of errorsRDD, warningRDD and outputRDD. An action operation is performed in order to get the count of the number of bad lines.

2.3. Spark Framework

The Spark framework consists of many closely integrated components and at its core, Spark is a computational engine which is responsible for scheduling, distributing and monitoring the tasks that have been distributed among the machines of the cluster [6]. The major components are:

- *Spark Core*: Spark Core contains the basic functionality of Spark and also manages the API that defines Resilient Distributed Datasets. The most important Spark abstraction is RDD [2]. Spark core consist of components for task scheduling, memory management, fault recovery, interacting with storage systems and more.
- *Spark SQL*: Interaction with Spark is possible with the help of SQL and Apache Hive-variant of SQL, called the Hive Query Language. Spark SQL represents database tables as Spark RDD's and provides an SQL interface to spark. It also translates SQL queries into Spark operations. This component was added in version 1.0.
- *Spark Streaming*: One of the biggest advantage of Spark over Hadoop is its ability to deal with streaming data. Spark Streaming is a component that enables processing live streams of data. Spark Streaming runs a streaming computation as a series of small, deterministic batch jobs. The programming model that is being used is called discretized stream processing-DStream [11]. Each treats each batch of data as RDDs and processes them using RDD operations. Finally the processed results of the RDD operation are returned in batches. It divides the live stream into batches of X seconds as shown in Figure 2.
- *MLlib*: MLlib is another component of the Spark framework and provides multiple types of machine

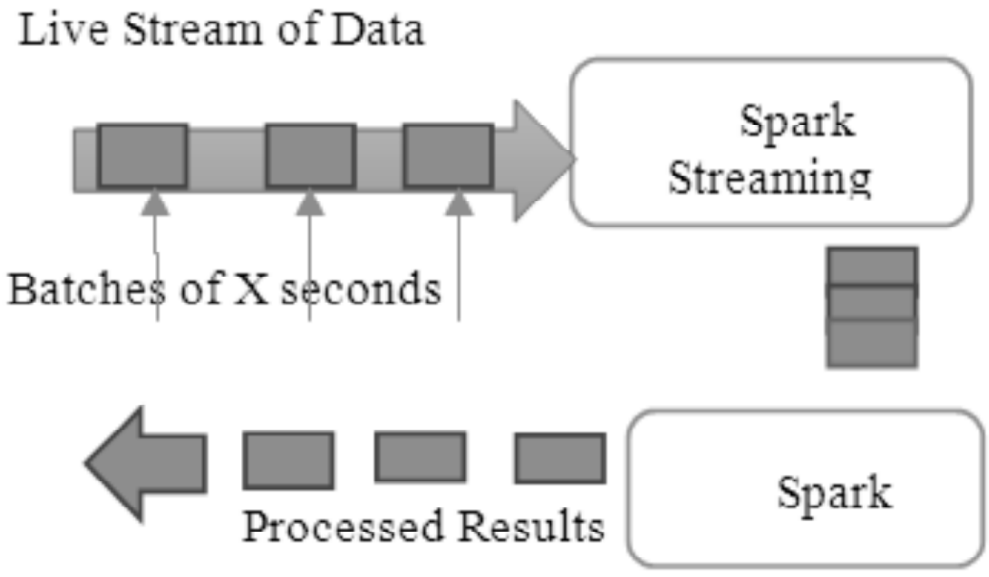

Figure 2: Spark Streaming

learning including binary classification, regression, clustering and collaborative filtering, as well as supports functionality such as model evaluation and data import.

- *GraphX*: Like Spark Streaming and MLlib, GraphX provides a collection of operators for manipulating graphs. It is a library added in Spark 0.9 and provides an API for computing and manipulating graphs.

2.4. Advantages of Spark over Hadoop

- *Faster*: Spark framework is 100 times faster than the Hadoop framework if the data is stored in the memory and 10 times faster if it is stored in the disk. Because Spark allows in-memory processing and sharing, execution of jobs are much faster.
- *Real time stream processing*: One of the biggest disadvantages of Hadoop is its inability to handle live stream of data. Spark can manage both batch processing as well as streaming jobs. Spark provides a Spark Streaming component in which it is possible to pass data through various software functions for performing data analytics.
- *Easy management***:** Organizations are able to do stream processing, graph processing, machine learning on a single cluster with the help of Spark.
- *Compatibility*: Apache Spark can run as standalone or on top of Hadoop YARN or Mesos on premise or on the cloud.
- *Ease of use*: Spark has comfortable APIs for Java, Scala and Python, and also includes Spark SQL (formerly known as Shark). Working with Hadoop is very complex but with Spark, we have the option of working with scala, which is very user friendly and easy to work with.

2.5. Scala

Scala integrates both functional programming features as well as object oriented features [3]. It is compiled to run on java virtual machine. Scala is functional, object oriented, runs on JVM, and is statically typed.

3. IMPLEMENTATION AND RESULTS

The Spark framework was setup on a single machine and a word count problem was executed. A word count example reads text files and counts how often words occur. It takes in a text file as input and the output is also a text file, where each line contains a word and the count of how often it occurred, separated by a tab. Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum which is depicted in Figure 3. Similarly Hadoop was setup in Ubuntu on a single machine and the same example was carried out. It has been represented in Figure 4. The word count example was carried out by taking different sizes of datasets and the time taken by both the frameworks have been tabulated in Table 1. It can be inferred that Spark performs the map reduce job faster than Hadoop. Table 2 summarizes the performance metrics in both Spark and Hadoop.

Table 1
Comparative performance of Wordcount Implementation

Dataset Sizes	*Hadoop*	*Spark*
500 MB	399.47s	41.121143s
1 GB	812.98s	83.348852s
3.5 GB	2417.49s	361.526s

Table 2
Summarization of features Spark Vs Hadoop

Performance Metrics	*Spark*	*Hadoop*	*Reason*
Real time stream processing	□√	⨻	Dstream model in Streaming component
Early termination	√	⨻	Modularizes computations so that no worker becomes idle after early termination
Iterative processing	√	⨻	RDDs can be cached for further map reduce tasks and data need not be read again and again
Faster	√	⨻	In-memory and non-redundant storage
Most active open source project in big data	√	√	In the previous year, Spark has the largest number of project contributors
Interactive processing	√	⨻	Spark consists of an interactive shell
Multiple APIs	√	⨻	Provides simple API in scala, python and java and R
Easy access to input data	√	⨻	Data is stored in the form of RDD in memory across the nodes
Easy management	√	⨻	Possible to perform streaming, batch processing and machine learning all in the same cluster
Lower latency computations	√	⨻	By caching the partial results across its memory of distributed computers
Fault-tolerance	√	√	Unlike Hadoop, Spark recovers its lost data through lineage
Compatibility	√	⨻	Compatible with Hadoop clusters, Mesos

4. CONCLUSION

Among all the frameworks which are currently available, Apache Spark is gaining popularity because of its unique features and simple approach. We speculate that in the near future most of the big data analytics will require real time semantics with no or very less need for housing the data hence the spark framework will play a major role.

References

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets", In proceedings of the 2nd USENIX Conference on Hot topics in cloud computing, pp. 10-10, June 2010.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. McCauley, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing", Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, pp. 2-2, USENIX Association, April 2012.

[3] T. Kumawat, P. K. Sharma, D. Verma, K. Joshi, V. Kumawat, and J. Vidyapeet, "Implementation of spark cluster technique with scala", International Journal of Scientific and Research Publications (IJSRP), 2(11), 2012.

[4] J. Dean, S.Ghemawat, S., "MapReduce: simplified data processing on large clusters", Communications of the ACM, 51(1), 107-113.Duan, A., May 2012.

[5] AiLing Duan, "Research and application of distributed parallel search hadoop algorithm", In Systems and Informatics (ICSAI), 2012 International Conference on Systems and Informatics (ICSAI 2012), IEEE, pp. 2462-2465.

[6] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia, "Learning Spark", O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[7] D. Borthakur, HDFS architecture guide, Hadoop Apache Project, 53, 2008.

[8] https://spark.apache.org/examples.html

[9] https://sankalplabs.wordpress.com/2014/08/25/installing-apache-spark-on-windows-step-by-step-approach/

[10] http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[11] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica,"Discretized streams: Fault-tolerant streaming computation at scale", In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (pp. 423-438), ACM, November 2013.